02222: Distributed Systems
Lab9: Grid Computations

Robin Sharp

April 2007

The aim of this lab is to give you some practice in the use of Grid technology. For a refreshing change from the PartFinder system, the application for this final lab exercise involves an (in principle) exceedingly large calculation, which you need to divide into small portions and submit to NorduGrid.

This exercise assumes that you have already carried out Lab8 (Grid Tutorial), so that you have installed the NorduGrid client software and obtained a valid certificate for authentication purposes.

1 System description

The application for this lab is to find the factors of a large integer by using the Quadratic Sieve (QS) algorithm. Although this is not the most efficient factorisation algorithm known today, it is relatively simple to implement using a master/slave system, where the master sets up a number of relatively small sub-tasks and sends them to the slaves. A parallel implementation of this type has been described by Cosnard and Philippe.

1.1 Mathematical basis

The mathematical basis of the QS algorithm is as follows: Suppose an integer N is to be factorised. We try to find two (different) integers, say y and z, such that $y^2 = z^2 \pmod N$ and $y \neq \pm z \pmod N$. If we can do this, we know that N does not divide $y + z$ or $y - z$, so $\gcd(N, y + z)$ and $\gcd(N, y - z)$ are proper factors of N.

Now let $m = \lfloor \sqrt{N} \rfloor$, and consider the polynomial $q(x) = (x + m)^2 - N$. It is then clear that:

$$q(x) = x^2 + 2mx + m^2 - N \approx x^2 + 2mx$$

which is small relative to N if x is small in absolute value. The QS algorithm selects $a_i = (x + m)$ and tests whether all the prime factors of $b_i = (x + m)^2 - N$ are less than some chosen (prime) bound, p_k. (b_i is then said to be *"p_k smooth"*.) Now

$$a_i^2 = (x + m)^2 \equiv b_i \pmod N$$

Furthermore, if prime p divides b_i, then

$$(x+m)^2 \equiv N \pmod{p}$$

so N is a quadratic residue modulo p. This means that it is only necessary to consider as possible factors of N those primes p for which the *Legendre symbol*, $(\frac{N}{p}) = 1$, where:

$$(\frac{N}{p}) = \begin{cases} 0 & \text{if } p|a \\ 1 & \text{if } a \in Q_p \\ -1 & \text{if } a \in \overline{Q_p} \end{cases}$$

Here Q_p is the set of quadratic residues modulo p. The set of odd primes for which this condition holds, together with the value -1, make up the *factor base* for use in the algorithm.

1.2 A parallelisable version of the QS algorithm

The steps in Cosnard and Philippe's version of the QS algorithm can be described as follows:

1. **Initialisation.**
   (a) Choose the number of elements k to be included in the factor base.
   (b) Choose the sieve interval $[-M, M]$
   (c) Calculate B, the smallest power of 2 greater than every element of the base.
   (d) Compute and store the k elements p_i in the factor base.
   (e) Compute and store the solutions of:
   $x^2 \equiv N \bmod {p_i}^\alpha$, with ${p_i}^\alpha \leq B$
2. **Compute polynomial.**
   The polynomials in Quadratic Sieve are used to generate integers that factor on the base. The polynomial is of the form:
   $w(x) = D^2 * x^2 + 2bx + c$
   (a) Compute D, which must be a prime:
   $D = 4k + 3$
   (b) From D we can deduce b and c:
   $b = N^{\frac{\frac{D(D-1)}{2}+1}{2}} \bmod D^2$ where $|b| < \frac{D^2}{2}$
   $c = \frac{b^2-N}{D^2}$
3. **Sieve.**
   The sieve step is the most time consuming step in the algorithm. Since we now have a set of primes for our factor base, we can calculate $w(x)$ by taking a number x from our sieving interval and checking if it factors completely over the factor base. If it factors we keep it, otherwise it is rejected.
   (a) Sieve with each element p_i of the base and each power α, such that $p_i^\alpha \leq B$
   (b) Initialise an array $tab_sieve[-M, M[$ to 0

   (c) For each ${p_i}^{\alpha}$, starting with an index x close to $-M$ and until $x \geq M$:
   Add $\log_2(p_i)$ to $tab_sieve[x]$ and set $x = x + {p_i}^{\alpha}$

4. **Define bound V.**
   V should be selected so that as little time as possible is spent on those $w(x)$ that do not factor on the base. For each x in the sieve interval do:
   (a) Compute $w(x)$.
   (b) Try to factor $w(x)$ over the base. If it succeeds, $w(x)$ should be stored as a row in matrix K.
   If $w(x)$ is not yet factored, go to step 2.
5. **Gauss elimination.**
   (a) Perform Gauss elimination on the matrix K containing all $w(x)$.
   (b) Compute the *gcd* of some dependent lines of the matrix K. The *gcd* is a cofactor of N.

Apart from the size of the integer N to be factorised, the most important parameters which affect the computational effort required are:

1. The number k of primes in the factor base.
2. The size of the sieve interval $[-M, +M[$.

The optimal value of k is given roughly by the formula:

$$k \approx \exp(0.5\sqrt{\ln N}\sqrt{\ln \ln N})$$

M needs to be reasonably large in order to ensure a high probability of finding a factor, but the effort required is roughly proportional to M. However, it is just exactly this feature which makes it sensible to parallelise the algorithm, as sieving can take place in parallel on several computers.

For more details about the Quadratic Sieve algorithm, you could for example look in Section 3.2.6 of *"Handbook of Applied Cryptography"* by Menezes, van Oorschot and Vanstone (CRC Press, 1997). However, in this lab, you will be supplied with most of the code required, so there should be no need to program the actual algorithm yourself.

1.3 A standalone version of the algorithm

If you want a "reference version" of the algorithm to compare your parallel version of the program with, there is a non-parallel, standalone version written in Java available for downloading from:

```
ftp://www6.software.ibm.com/software/developer/library/gr-factor/factorization.zip
```

Use your browser to download this zip-file, and then unzip it in a convenient directory. This will produce a new sub-directory `Factorization`, which contains a number of further sub-directories. Make `Factorization` your current directory, and then (in Linux) execute the *bash* command:

```
export CLASSPATH=$CLASSPATH:./classes
```

You can then run the standalone version of the factorisation program by typing:

```
java javax.math.factorization.Factorizer -n xxxxx
```

where xxxxx is the number you want to factorise. Some interesting numbers to try out are:

Name	Number
T_{20}	18567078082619935259
T_{30}	350243405507562291174415825999
T_{40}	5705979550618670446308578858542675373983
T_{45}	732197471686198597184965476425281169401188191
T_{50}	53468946676763197941455249471721044636943883361749
T_{55}	5945326581537513157038636316967257854322393895035230547
T_{60}	676292275716558246502605230897191366469551764092181362779759

Note that T_i consists of i decimal digits. If you try out these numbers in increasing order, you will find that the computation time increases noticeably from the smallest to the largest. For really big numbers, a non-parallel Java program is not a practical solution.

1.4 Organising the parallel algorithm

In the parallel version of the QS algorithm, a master process communicates values out to a number of slaves on request, and receives results from them. A possible organisation of the master (or "server") and slaves ("clients") is shown in Figure 1.

The master and slaves communicate by exchanging messages. Roughly speaking, each slave requests the master to send it the (current) value of N and a initial D-value. If the value of N is new, the slave evaluates an appropriate new factor base; otherwise it continues to use the current one. The slave then creates a polynomial of the form $w(x) = D^2x^2 + 2bx + c$, initialises it and sieves it to find any results. The slave then evaluates a new D-value from the previous one, and looks for any results. This procedure is repeated until a given number of D-values have been considered, and the slave then sends all the results to the master, who stores them in the result matrix. When the master has received enough results from the slaves to complete the matrix, it performs the Gaussian elimination step of the algorithm, and inspects to see whether this reveals a non-trivial factor (i.e. one which is not 1 or N).

2 Implementation

There are two aspects to implementing a Grid-based application:

1. Producing code for the various sub-tasks.
2. Deploying the code among the computers to which you have access.

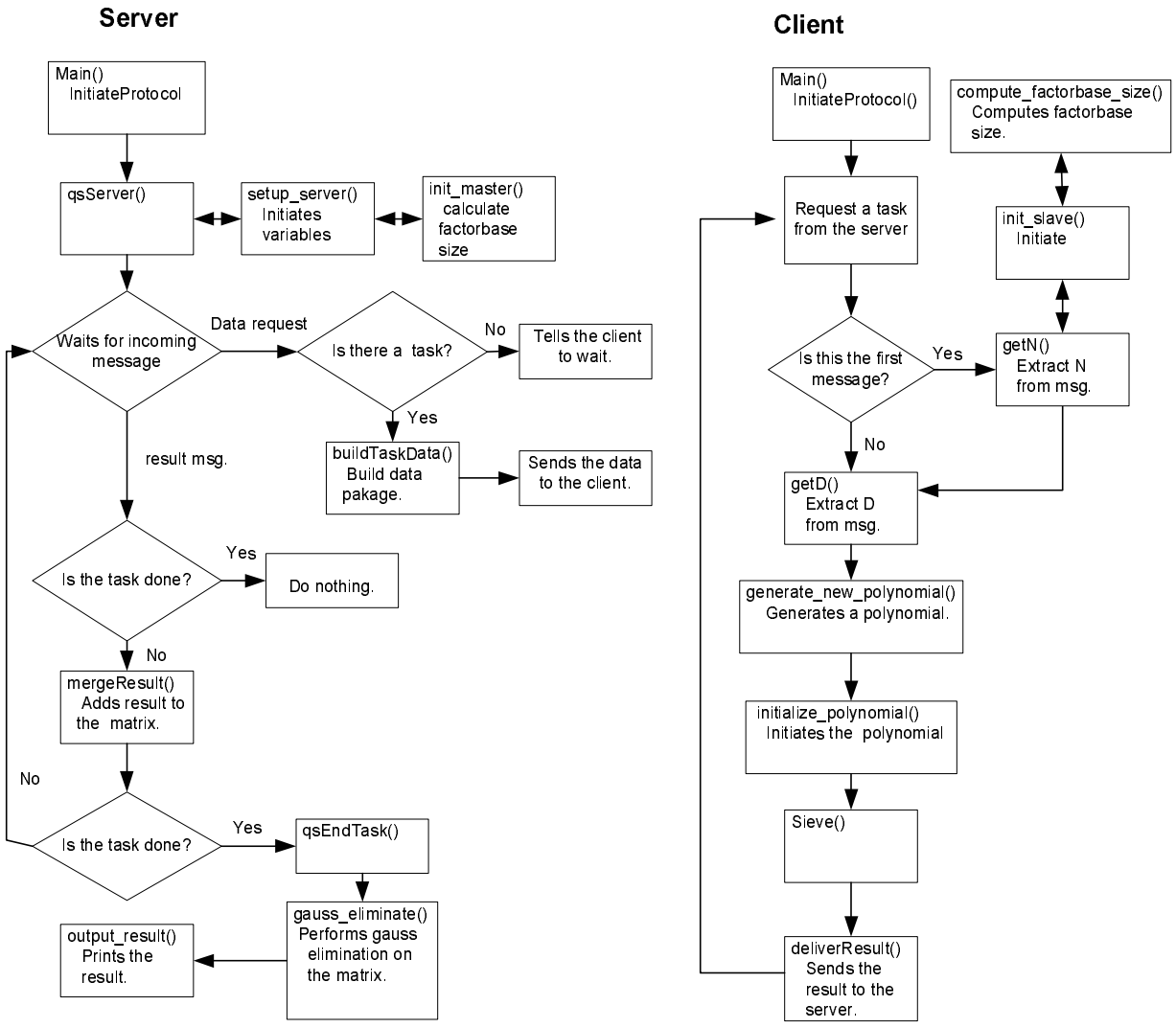

Figure 1: Application Flow

2.1 Producing the code

The calculations for this application require the use of a package which can handle very large integers. The package recommended for this purpose in connection with the software discussed below is GMP, available from:

`http://www.swox.com/gmp`

You should use version 4.1.2 (or later) of the package. The most recent version is currently 4.1.4. If you are working in the E-databar, you will find that GMP is already installed.

The remainder of the code is available as two compressed tar-balls which can be retrieved from:

`http://www.imm.dtu.dk/~robin/02222/grid/`

The tar-ball `qGridClient.tgz` contains the code to be run on a slave (client), and `qGridServer.tgz` the code for the master (server). Each of these tar-balls contains an appropriate Makefile. You should unpack them and compile them on your machine.

2.2 The results database

The server/master code requires the presence of a MySQL system on the computer on which it is to run, in order to set up a database for storing partial results. The MySQL client library is already installed on the computers in the E-databar, but if you are working elsewhere and you do not have MySQL, it is available as a freeware distribution from:

```
http://www.mysql.com
```

Download the distribution and set it up by going to the root of the distribution and executing the commands:

```
cd setup
mysql -uroot -p < mysqlinit.sql
```

In the E-databar, there is a MySQL server, and an account has been set up on this server for each of the students who are registered for the course. To use your account, you need a userid (the same as your DTU student number) and a special password (i.e. not the usual databar password!). Each account is associated with a database area on the MySQL server, so essentially each student has access to a private database for use in this lab. The name of this database is the same as your student number. You can find your password in the directory MySQL in the File Sharing area for course 02222 in CampusNet. Be careful to use your own password!

Before trying to run the qGrid server, you will need to set up the two Mysql tables ('global' and 'pending') which it uses. Suitable SQL commands to do this are:

```
CREATE TABLE global (name VARCHAR(20),
                     value INT);

CREATE TABLE pending(packageid INT,
                     lastclientid VARCHAR(20),
                     taskid INT,
                     progress INT,
                     lastupdate DATETIME,
                     data MEDIUMBLOB);
```

(The data column in `pending` actually has a maximum length of MAX_SIZE_OF_MESS, which is equal to 263579 bytes.) You will also need to initialise the values of the `clientidcounter`, `packageidcounter` and `taskidcounter` fields in the `global` table. Initial values of 0 are suitable.

Finally, you will have to modify the file `mysql_util.h`, which contains a number of definitions related to the database. The things which have to be set are:

MY_PASSWORD: Set this to your password on the MySQL server.
MY_USER: Set this to your user id on the MySQL server.
MY_HOST: Set this to '192.38.93.11', the IP address of the MySQL server.
MY_DATABASE: Set this to the name of your database. (As stated above, this should be the same as your student number.)

The MySQL server uses the MySQL default port number 3306. When you have edited `mysql_util.h`, remember to recompile the qGridServer software before using it!

2.3 Deploying the master and slave code

In a Grid-based solution, the sub-tasks to be run on the slaves will just be submitted to the Grid, where the administrative programs will find suitable computers on which they can be run. Roughly speaking, a "suitable computer" is one which fulfils two requirements:

1. It must have resources available which fulfil the requirements of one of the sub-tasks.
2. The originator of the application must be authorised to run on the computer concerned.

You will need to read the *NorduGrid User Manual*, available at:

`http://www.nordugrid.org/documents/userguide.pdf`

in order to find out now to describe tasks and send them out onto NorduGrid. You may also find it useful to look in the NorduGrid toolkit user interface manual, at:

`http://www.nordugrid.org/documents/NorduGrid-UI.pdf`

and the detailed manual for the XRSL job description language at:

`http://www.nordugrid.org/documents/xrsl.pdf`

You should remember that if you have compiled the software on a machine with a particular architecture and particular software requirements, then you will need to specify that these requirements must be fulfilled on the machines on which the application is to run.

You may be wondering how standard input and output are dealt with. A feature of most Grid computing environments (including the NorduGrid/ARC environment used here) is that standard input (stdin) for programs running on each individual node needs to be read from a file on that node, and that standard and error outputs (stdout, stderr) are likewise sent to a file on the local node. You can use the facilities described in Chapters 8 and 9 of the User Manual to move these files around, so you for example, after executing a job, can fetch the files with standard output in order to read them on the computer on which you are working.

In the software as provided, the number of the communication port used by the master and slaves, the values of N and the number of D-values to be dealt with on each activation of a slave are all specified in the code of `qGrid.c` in the master/server. Likewise, the code of `qGrid.c` in the slave/client specifies the IP address of the master/server. You

should consider how this needs to be changed, so that NorduGrid's facilities for specifying run-time parameters for applications are used instead.

In a Grid system, the participating computers will often be grouped in clusters which use non-public IP addresses internally within the cluster. This means that it is not in general possible to set up a TCP connection *to* the slaves which are distributed round and about in the Grid. In the software given here, you should notice that it is always the slaves/clients which initiate communication with the master/server, never the other way round. Of course, this strategy only resolves the difficulty if:

1. The master is running on a computer which has a public IP address and accepts connections on the specified communication port.
2. The slaves are running on nodes which allow outgoing connections to be set up via the specified communication port.

The simplest way to do this is to execute the master/server on a computer in the E-databar (or at home), and to specify that the slaves must run on Grid nodes which allow the attribute *nodeAccess* to have the value *outbound* (see the XRSL manual, page 22). Make sure also to choose a TCP port on which traffic can pass the firewalls which it meets on its route. You need to make sure that the computer on which the qGrid server is running is set up to accept incoming TCP connections in the range 40000-40040. This can be done by setting the Globus parameter `GLOBUS_TCP_PORT_RANGE`, for example by using the bash command:

```
export GLOBUS_TCP_PORT_RANGE=40000,40040
```

3 Report requirements

This lab is a mandatory part of the course, which means that you are expected to hand in a small report which will be evaluated and count towards your final grade. You are expected to work in groups of 2 participants. Your report should document your experiences with using the factorisation algorithm, especially in the parallel version. You should put most emphasis on aspects of your solution which are particular to the use of the Grid as implementation technology. You are particularly encouraged to investigate how the time required to execute the algorithm depends on the size of the number to be factorised and the various parameters of the algorithm such as k, M and the number of slaves. The report should be limited to a *maximum* of 5 pages, excluding the source code.

The report should be handed in to one of the mailboxes marked **02222** in the entrance at the West end of Building 322, not later than 16:00 on **Wednesday 9 May** (the last day of the 13-week teaching period).